日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

実現可能性判定コスト削減のための無限ゲーム単純化手法

島川 昌也　萩原 茂樹　米崎 直樹

リアクティブシステム仕様の実現可能性に関する検証は，仕様の実現システム集合を表現する無限木オートマトンあるいは有限グラフ上の無限ゲームを構成し，それを分析することで行われる．しかし，それらのサイズは検証対象の仕様のサイズに対して爆発的に大きくなり，その構成・分析の計算コストは極めて高い．本研究では，実現可能性判定コスト削減のための無限ゲームの単純化手法を与える．実現システムは無限ゲームに対する必勝戦略に対応し，ここで与える単純化は，必勝戦略の存在を変化させない単純化であり，通常行われるそれが表現するシステム集合自体を変化させない単純化に比べて強力である．本研究では，無限ゲームの構成自体のコストを下げるため，その単純化をゲームの構成途中に適用する方法を与え，その実験結果についても報告する．

1 はじめに

システムの仕様段階での分析は，開発プロセス全体を通じてのコスト削減の観点からは，作成後のシステムの検証よりも重要であるといわれている [10]．環境からの要求に対して適切なタイミングで応答を返すことが求められるリアクティブシステムを対象とした場合，動作仕様としてその可能な振る舞い集合を定める制約を記述し，分析することが有効である．このようなシステムの仕様に求められることは，単に無矛盾である（仕様を満たす振る舞いが存在する）ことだけではない．環境に対して開いたリアクティブシステムでは，その振る舞いは環境の振る舞いに依存する．また，環境の振る舞いをシステムはコントロールすることができない．そのため，リアクティブシステム仕様は,「環境からどのような要求イベントがどのような順序で発せられても，仕様を満たすような応答を過去の要求履歴より決定できる」という実現可能性 [13] [1] を満たすことが望まれる．実現可能性の検査を行うことで，仕様に潜む見えにくい欠陥を検出することが可能となる．さらに，実現可能であることが分かれば，原理的にプログラム合成が可能である [13]．すなわち，有限状態機械で表現される実現システム（そのすべての振る舞いが仕様を満たすことが保障されているシステム）を得ることができる．

リアクティブシステム仕様の実現可能性に関する検証やプログラム合成は，その必勝戦略が仕様を満たす実現システムと対応する有限グラフをたどり続ける無限ゲーム，もしくはそれが受理する無限木が実現システムと対応する無限木オートマトンを構成し，それを分析することで行われる [13] [1] [12] [7]．しかし，その無限木オートマトンや無限ゲームの表現のサイズは検証対象の仕様のサイズに対して爆発的（仕様を線形時間論理 LTL で記述した場合，2 重指数）に大きくなり，その構成・分析の計算コストは極めて高い．

そこで本研究では，実現可能性判定コスト削減のための無限ゲームの単純化手法を与える．ここで与える単純化は，無限ゲームの模倣関係を基に，必勝戦略が存在するかを調べる上で余分な，無限ゲームを行う有限グラフのエッジを除去するものである．直感的に

A simplification method for infinite games to reduce the cost of realizability checking.

Masaya Shimakawa, Shigeki Hagihara, Naoki Yonezaki , 東京工業大学大学院情報理工学研究科計算工学専攻, Dept. of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology.

は，意味的に連言（選言）の分岐に他より制約の弱い（強い）ノードへのエッジがあった場合，それを除去するというものである．この単純化は，無限ゲームを行うグラフの構成途中において適用することが可能である．これは，グラフ構成時において各ノードにラベルされる情報を基に，構成されるゲームの模倣関係が得られるためである．それゆえ，無限ゲームを行うグラフの構成自体のコストを下げることができる．

必勝戦略は仕様を満たす実現システムと対応するため，本研究で与える無限ゲームの単純化は，「実現システムが存在するか」を変化させない単純化であるといえる．実現可能性判定においては，「実現システムが存在するか」が変化しないという条件さえ満たせばよい．[11] で提案されている「それが表現する実現システム集合」自体を変化させない単純化と比べて緩い条件を基にするため，本単純化はより強力である．

我々は，本研究で与える無限ゲームの単純化を行う実現可能性判定とそれを行わないものの性能を比較する実験を行い，本単純化を行うことで，より効率的に実現可能性判定を行えることを確かめたことを報告する．

2 リアクティブシステム仕様の実現可能性

2.1 リアクティブシステム

リアクティブシステムとは，環境とのインタラクションの維持を目的とするシステムであり，$RS = (X, Y, r)$ と形式化できる．ここで，

- X は環境が生起する要求イベントの集合である．
- Y はシステムが生起する応答イベントの集合である．
- $r : (2^X)^+ \to 2^Y$ は，リアクション関数である．リアクション関数とは，以前に生起した要求イベントの集合の列から現時点においてシステムが生起する応答イベントの集合を決定する関数である．

リアクティブシステムは以下のように環境とインタラクションを行う．リアクティブシステムは，はじめに環境が生起させた要求イベントの集合が $a_0 \in 2^X$ であるとすると，その時点で $r(a_0)$ に含まれる応答イベントを生起させる．次に環境が $a_1 \in 2^X$ に含まれる要求イベントを生起させると，$r(a_0a_1)$ に含まれる応答イベントを生起させる．以下同様に，時点 i に環境が $a_i \in 2^X$ に含まれる要求イベントを生起させると，リアクティブシステムは $r(a_0a_1 \dots a_i)$ に含まれる応答イベントを生起させていく．

環境が生起させる要求イベント集合の列 $\tilde{a} = a_0a_1a_2 \dots$ に対するリアクティブシステム $RS = (X, Y, r)$ の振る舞いを

$$(a_0 \cup r(a_0))(a_1 \cup r(a_0a_1))(a_2 \cup r(a_0a_1a_2)) \dots$$

とし，$behave_{RS}(\tilde{a})$ と表記する．

2.2 リアクティブシステムの動作仕様

リアクティブシステムの動作仕様は，システムの可能な無限長の振る舞いの集合を定める．本稿では，リアクティブシステムの仕様記述言語として線形時間論理（Linear Temporal Logic, 以下 LTL）を用いる．

本稿で扱う LTL は，時間演算子として $\mathbf{X}$（neXt 演算子），$\mathbf{U}$（Until 演算子）と $\mathbf{R}$（Release 演算子）を持つものであり，原子命題として要求イベントと応答イベントを用いる．要求イベントの集合 X と応答イベントの集合 Y が与えらたとき，LTL 式は以下のように定義される．

$$f ::= p \mid \neg f \mid f \wedge f \mid f \vee f \mid \mathbf{X}f \mid f\mathbf{U}f \mid f\mathbf{R}f.$$

ここで，$p \in X \cup Y$ とする．

LTL 式は，振る舞い, つまり $X \cup Y$ の部分集合の無限列上に解釈される．f を LTL 式とし，$\sigma \in (2^{X \cup Y})^\omega$ を振る舞いとする．σ が f を満たすことを $\sigma \models f$ と表し，以下のように再帰的に定義する．ここで，$\sigma[i]$ は σ の i 番目の要素を，$\sigma[i \dots]$ は σ の i 番目以降の振る舞いを表す．

- $\sigma \models p$ **iff** $p \in \sigma[0]$,
- $\sigma \models \neg f$ **iff** $\sigma \not\models f$,
- $\sigma \models f_1 \wedge f_2$ **iff** $\sigma \models f_1$ and $\sigma \models f_2$,
- $\sigma \models f_1 \vee f_2$ **iff** $\sigma \models f_1$ or $\sigma \models f_2$,
- $\sigma \models \mathbf{X}f$ **iff** $\sigma[1 \dots] \models f$,
- $\sigma \models f_1\mathbf{U}f_2$ **iff** $\exists j (j \geq 0).$ $\Big($ $\sigma[j \dots] \models f_2$ and $\forall k (0 \leq k < j).\ \sigma[k \dots] \models f_1 \Big)$,
- $\sigma \models f_1\mathbf{R}f_2$ **iff** $\forall j (j \geq 0).$ $\Big($ $\sigma[j \dots] \models f_2$ or $\exists k (0 \leq k < j).\ \sigma[k \dots] \models f_1 \Big)$.

2.3 実現可能性

リアクティブシステム仕様が実現可能であるとは，「どのような要求イベントがどのような順序で生起しても，仕様を満たすように応答できるようなリアクティブシステムが存在する」という性質であり，次のように形式化される．

ψ を LTL によるリアクティブシステム仕様とする．以下を満たすリアクティブシステム RS を ψ の実現という．

$$\forall \tilde{a} \in (2^X)^\omega . behave_{RS}(\tilde{a}) \models \psi$$

そのような実現が存在するとき，ψ は実現可能であるとする．

3 実現可能性判定

本章では，実現可能性判定法について説明する．実現可能性判定は，通常次のような手順で行われる．

1. 仕様の振る舞いを受理する無限語上のオートマトン $\mathcal{A}_{bhv}$ を構成.
2. $\mathcal{A}_{bhv}$ を基に，その必勝戦略が仕様の実現システムと対応する有限グラフをたどり続ける無限ゲーム $\mathcal{G}_{sys}$ を構成する†1．もしくは，その受理無限木が実現システムと対応する無限木上のオートマトン $\mathcal{A}_{sys}$ を構成する.
3. $\mathcal{G}_{sys}$ に必勝戦略が存在するか，もしくは $\mathcal{A}_{sys}$ に受理木が存在するかを調べる.
   - 存在するならば,「実現可能である」と判定.
   - 存在しないならば,「実現可能でない」と判定.

実現可能性判定法は，Safra の決定化 [14] を必要とするもの（[13] [1] 等）と，必要としないもの（[12] [15] [7] 等）とに大別されるが，本章では，後者の [15] [7] の判定法を基にした無限ゲームを用いる実現可能性判定法を紹介する.

†1 環境が要求を発することをゲームにおいて敵が動作することに，そして，環境とシステムによる振る舞いが仕様を満たすことをゲームのプレイにおいて敵に勝利することに対応させると，環境からの任意の要求に対して仕様を満たす実現システムはゲームにおける敵がどのように振る舞っても勝利する必勝戦略と対応する.

3.1 準備：ω オートマトン

無限語上のオートマトン．無限語上のオートマトン $\mathcal{A} = (\Sigma, Q, q_I, \delta, F)$ は，Σ : アルファベット，Q : 有限の状態の集合，$q_I \in Q$: 初期状態，$\delta \subseteq Q \times \Sigma \times Q$: 遷移関係，F : 終了状態の集合より構成される．Σ 上の語 σ に対する行程は，状態の列 $\varrho \in Q^\omega$ で，$\varrho[0] = q_I$，かつすべての $i \geq 0$ において $(\varrho[k], \sigma[k], \varrho[k+1]) \in \delta$ となるものである．語 σ に対する $\mathcal{A}$ に対する行程の集合 $Runs_{\mathcal{A}}(\sigma)$ で表す．$Runs_{\mathcal{A}}(\sigma)$ が以下に定義する受理条件を満たすとき，$\mathcal{A}$ は σ を受理するという．

受理条件．本稿では，受理条件として，以下の非決定性 Büchi(NB)，ユニバーサル co-Büchi(UC)，ユニバーサル K-co-Büchi(UKC)，非決定性 Safety(Safety) の条件を用いる．

NB の条件:

$$\exists \varrho \in Runs_{\mathcal{A}}(\sigma). Visit(\varrho, F) = \infty$$

UC の条件：

$$\forall \varrho \in Runs_{\mathcal{A}}(\sigma). Visit(\varrho, F) < \infty$$

UKC の条件：K は自然数とする．

$$\forall \varrho \in Runs_{\mathcal{A}}(\sigma). Visit(\varrho, F) \leq K$$

Safety の条件：

$$\exists \varrho \in Runs_{\mathcal{A}}(\sigma). Occ(\varrho) \subseteq F$$

ここで，$Visit(\varrho, F)$ は ϱ が F に含まれる状態を通過する回数を表し，$Occ(\varrho)$ は ϱ が通過する状態の集合を表す.

$\mathcal{A}$ が受理条件 NB（UC, UKC, Safety）で受理する語の集合を $L_{\mathrm{nb}}(\mathcal{A})$（$L_{\mathrm{uc}}(\mathcal{A})$, $L_{\mathrm{uc},K}(\mathcal{A})$, $L_{\mathrm{saf}}(\mathcal{A})$）で表す．各受理条件 NB，UC，U$K$C による受理語集合について，以下が成立する．

- $L_{\mathrm{nb}}(\mathcal{A}) = \overline{L_{\mathrm{uc}}(\mathcal{A})}$.
- $0 \leq K_1 \leq K_2$ のとき，$L_{\mathrm{uc},K_1}(\mathcal{A}) \subseteq L_{\mathrm{uc},K_2}(\mathcal{A}) \subseteq L_{\mathrm{uc}}(\mathcal{A})$

オートマトンがすべての $q \in Q$,$a \in \Sigma$ に対して $(q, a, q') \in \delta$ となる q' が高々 1 つしか存在しないとき，そのオートマトンは 決定的であるといい，そのオートマトンを決定性オートマトンと呼ぶ.

3.2 準備：無限ゲーム

本研究では，有向グラフ上の 2 プレイヤの無限ゲームを扱う.

ここで扱うゲームは，2 種のノード（0-ノードと 1-ノード）を持つグラフの上で行われる．直観的には，このゲームは以下のように進行する．ある初期ノード v_0 にトークンが置かれており，そのノード v_0 が，0-ノードのときはプレイヤ 0 が，1-ノードのときはプレイヤ 1 が，そのノードに隣接する次ノード v_1 をひとつ選択し，そこにトークンを移動する．移動された先 v_1 でも同様に，トークンのあるノード v_1 の種類に応じたプレイヤが隣接するノード v_2 を選択し，トークンを移動する．これを繰り返す．ゲームの勝者は，トークンが通過したノードの列が予め定めておいた勝利条件（ω オートマトンと受理条件と同様な方法で定められる）を満たすかによって決定される．

このような無限ゲームの形式的な定義を以下に与える．

無限ゲーム．無限ゲーム $\mathcal{G} = (V_0, V_1, E, Win)$ は，V_0:0-ノードの有限集合，V_1:1-ノードの有限集合，$E \subseteq (V_0 \cup V_1) \times (V_0 \cup V_1)$: エッジの集合，Win : 勝利条件からなる．$V_0 \cup V_1$ を V で表す．ノード v の次ノード集合を $vE = \{v' \in V | (v, v') \in E\}$ と定義する．本稿では，$E \subseteq (V_0 \times V_1) \cup (V_0 \times V_1)$，すなわち二部グラフであること，及び，任意の v で $vE \neq \emptyset$ である，すなわち行き止まりが存在しないことを仮定する†2．

プレイ．ゲーム $\mathcal{G}$ のプレイは，無限パス $\pi = v_0 v_1 \ldots \in V^\omega$ であり，任意の $i \in \mathbb{N}$ において $v_{i+1} \in v_i E$ であるものである．

プレイ π が勝利条件を満たすときプレイヤ 0 の勝利であるとする．本稿では，勝利条件として，無限語上のオートマトンの受理条件と同様の Safety の条件を用いる．すなわち，$Win = F \subseteq V$ であり，

$$Occ(\pi) \subseteq F.$$

を満たすとき，プレイ π の勝者はプレイヤ 0 である．勝利条件として Safety の条件を用いるゲームを Safety ゲームと呼ぶ．

戦略．プレイヤ 0 の戦略は，部分関数 $f : V^* V_0 \to V$ であり，$f(\pi v)$ が定義されているときは必ず $f(\pi v) \in vE$ であるものである．プレイ $\pi = v_0 v_1 \ldots$ が，すべての $v_i \in V_0$ において，$f(v_0 v_1 \ldots v_i)$ が定義されているとき $v_{i+1} \in f(v_0 v_1 \ldots v_i)$ である場合，プレイ $\pi = v_0 v_1 \ldots$ は f に従っているという．ゲーム $\mathcal{G}$ の f に従うノード v より始まるプレイの集合を $play_{\mathcal{G}}(f, v)$ で表す．戦略 f は，任意のプレイ $\pi \in play_{\mathcal{G}}(f, v)$ がプレイヤ 0 の勝利であるとき，v において必勝であるという．v において必勝である戦略が存在するとき，プレイヤ 0 は v で必勝であるという．

3.3 実現可能性判定

入力 : 仕様 ψ.

出力 :「実現可能である」 or 「実現可能でない」.

1. ψ より，$L_{\mathrm{uc}}(\mathcal{A}) = \{\sigma | \sigma \models \psi\}$ となるオートマトン $\mathcal{A} = (2^{X \cup Y}, Q, q_I, \delta, F)$ を構成する．K を自然数とする．このオートマトンは，$L_{\mathrm{uc},K}(\mathcal{A}) \subseteq L_{\mathrm{uc}}(\mathcal{A})$ であり，また，K を十分に大きくすると，「$L_{\mathrm{uc}}(\mathcal{A})$ が実現可能であるならば，かつそのときに限り，$L_{\mathrm{uc},K}(\mathcal{A})$ も実現可能である」を満たす．
2. $\mathcal{A}$ を決定化し，$L_{\mathrm{saf}}(\mathcal{D}) = L_{\mathrm{uc},K}(\mathcal{A})$ となる決定性オートマトン $\mathcal{D} = (2^{X \cup Y}, S, s_I, \Delta, F^{\mathrm{saf}})$ を構成する．
   - $S := \{s : Q \to [0 \ldots K, \infty]\}$.
   - $s_I : s_I(q_I) = K$, 任意の $q \in Q \setminus \{q_I\}$ において $s_I(q) = \infty$ である $s_I \in S$.
   - $F^{\mathrm{saf}} := \{s \mid \forall q \in F.\ s(q) > 0\}$.
   - $\Delta(s, a)$:
     - $s \notin F^{\mathrm{saf}}$ とき，s.
     - その他のとき，以下を満たす $s' \in S$.
       $s'(q) = min\{next(p) | (p, a, q) \in \delta\}$.
       ここで，$next(p)$ = if $q \in F$ then $s(p) - 1$ else $s(p)$.
       ただし，$min\{\emptyset\} = \infty$ とする．
3. $\mathcal{D}$ を Safety ゲーム $\mathcal{G} = (V_0, V_1, E, W)$ に変換する．
   - $V_1 := S$.
   - $V_0 := 2^S$.

†2 後で与える実現可能性においては，このようなゲームのみを扱うので，簡単のため，このような仮定をおく．しかし，本稿で与える単純化はこれらの仮定をおかないゲームにおいても同様に適用可能である．

- $E := \{(s, suc(s,x))|s \in V_1, x \in 2^X\} \cup \{(v_0, s)|v_0 \in V_0, s \in v_0\}$.
  ここで，$suc(s,x) := \{s'|\exists y \in 2^Y.(s, x \cup y, s') \in \Delta\}$.
- $W = F^{\mathrm{saf}} \cup V_0$

4. $\mathcal{G}$ が $s_I \in V_1$ で必勝であるか調べる．
   - 必勝であるとき,「実現可能である」を出力.
   - 必勝でないとき,「実現可能でない」を出力.

ステップ 1 の $L_{\mathrm{uc}}(\mathcal{A}) = \{\sigma|\sigma \models \psi\}$ であるオートマトン $\mathcal{A}$ の構成は，[9] 等の LTL から非決定性 Büchi オートマトンへの変換法を基に行える．$L_{\mathrm{uc}}(\mathcal{A}) = \overline{L_{\mathrm{nb}}(\mathcal{A})}$ であるため，その変換法を基に $L_{\mathrm{nb}}(\mathcal{A}) = \{\sigma|\sigma \models \neg\psi\}$ である $\mathcal{A}$ を構成すればよい．

ステップ 2 の決定性オートマトン構成法は，有限語上のオートマトンの決定化に利用される部分集合法を，決定化前の各状態について自然数を保持するように，拡張したものである．ここで構成するオートマトンの各状態 $s : Q \to [0 \ldots K, \infty]$ は，直感的には決定化前の各状態においてそこから残り何回 F に含まれる状態を通過していよいかを表している（∞ はその状態に到達不能であることを意味する）．

ステップ 3 の決定性オートマトンからゲームへの変換では，図 1 のように，決定性オートマトンの遷移を要求（2^X の要素）による遷移と応答（2^Y の要素）による遷移に分けて考え，要求による遷移を 1-ノードからのエッジに，応答による遷移を 1-ノードからのエッジに対応させている．決定性オートマトンの状態をゲームにおける 1-ノードとし，要求による遷移と応答による遷移の間に中間状態を用意し，それをゲームにおける 0-ノードとしている．中間状態はそこから応答によって遷移可能な状態の集合をラベルしている．決定性オートマトンに受理される（仕様を満たす）振る舞いによる行程をゲームにおける勝利プレイに対応させている．そうすることで，環境からの任意の要求に対して仕様を満たすように応答する実現システムはゲームにおけるプレイヤ 1 がどのように振る舞っても勝利するプレイヤ 0 の必勝戦略と対応する．それゆれ，そのゲームに必勝戦略に存在するかを調べることで，仕様に実現システムが存在するか（実現可能であるか）を判定できる．

4 無限ゲームの模倣関係による単純化

本章では，無限ゲームの模倣関係を導入し，それを基にしたゲームの単純化手法を与える．

4.1 模倣関係とその性質

ここでは，無限ゲームの模倣関係，およびその性質について与える．

定義 4.1 (Safety ゲームの模倣関係)**.** $\mathcal{G} = (V_0, V_1, E, W)$ と $\mathcal{G}' = (V_0', V_1', E', W')$ をそれぞれ *Safty* ゲームとする．以下の条件を満たす関係 $\preceq \subseteq (V_0 \times V_0') \cup (V_1 \times V_1')$ を $\mathcal{G}$ と $\mathcal{G}'$ との間の模倣関係という．

$v \preceq v' \Longrightarrow$

- $v \in W \Longrightarrow v' \in W'$, かつ
- $v \in V_0, v' \in V_0'$ のとき，$\forall u \in vE.\exists u' \in v'E'.u \preceq u'$,
  $v \in V_1, v' \in V_1'$ のとき，$\forall u' \in v'E'.\exists u \in vE.u \preceq u'$.

また，$\mathcal{G}' = \mathcal{G}$ として上の条件を満たす $\mathcal{G}$ のノード上の関係 $\preceq \subseteq (V_0 \times V_0) \cup (V_1 \times V_1)$ を $\mathcal{G}$ 上の模倣関係という．

定理 4.2. $\mathcal{G} = (V_0, V_1, E, W)$ と $\mathcal{G}' = (V_0', V_1', E', W')$ とをそれぞれゲームとする．また，$\preceq$ を $\mathcal{G}$ と $\mathcal{G}'$ との間の模倣関係とし，$v \preceq v'$ とする．このとき，$\mathcal{G}$ でのプレイヤ *0* の任意の戦略 f に対して，以下を満たす $\mathcal{G}'$ での戦略 f' が存在する．

$\forall \pi' \in play_{\mathcal{G}'}(f', v').\exists \pi \in play_{\mathcal{G}}(f, v).\pi \preceq_{sq} \pi'$.

ここで，$\pi \preceq_{sq} \pi'$ は，任意の i で $\pi[i] \preceq \pi'[i]$ であることを表す.

Proof. f を $\mathcal{G}$ でのプレイヤ 0 の戦略とする．次のように定義される f' を考える．

$f'(\pi' u') =$

- $\pi u \preceq^s_{sq} \pi' u'$ である $\mathcal{G}$ のパス πu が存在するとき，
  - $f(\pi u)$ が定義されているのとき，$f(\pi u) \preceq w'$ であるような $w' \in u'E'$.
  - $f(\pi u)$ が定義されていないとき，$w \preceq w'$ である $w \in uE$ が存在するような $w' \in u'E'$.
- その他のとき，未定義.

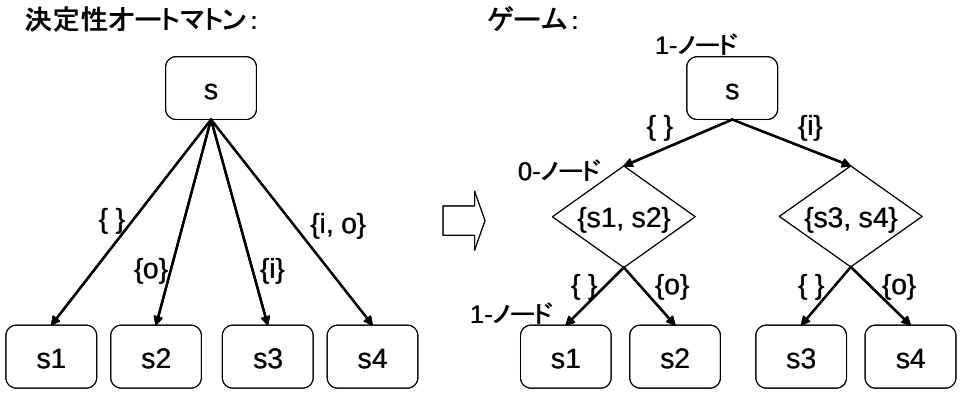

図 1　決定性オートマトンの遷移とゲームのエッジの対応（$X = \{i\}, Y = \{o\}$）

ここで，$\pi \preceq^{s}_{sq} \pi'$ は，$|\pi| = |\pi'|$ であり，任意の i で $\pi[i] \preceq \pi'[i]$ であることを表す.

$play_{\mathcal{G}'}(f', v')$ のすべてのプレイ π' について，$\pi \preceq_{sq} \pi'$ であり，$\pi \in play_{\mathcal{G}}(f, v)$ である π が存在することを示す．$\pi' \in play_{\mathcal{G}'}(f', v')$ とし，条件を満たす π の構成方法を与える.

- $\pi[0] = v$ とする．このとき，$\pi'[0] = v'$ であり，また，仮定より $v \preceq v'$ であるため，$\pi[0] \preceq \pi'[0]$ である.
- $\pi[i+1]$: 各 $0 \leq j \leq i+1$ において $\pi[j] \preceq \pi'[j]$ であることを仮定し，$\pi[i+1] \preceq \pi'[i+1]$ となる π の $i+1$ 番目のノードを以下に示す.
  - $\pi'[i] \in V_0$ の場合．$\pi[0 \ldots i] \preceq^{s}_{sq} \pi'[0 \ldots i]$ であるため，f' の定義より，$f'(\pi'[0 \ldots i])$ は未定義ではなく，$\pi'[i+1] = f'(\pi'[0 \ldots i])$ である.

    - $f(\pi[0 \ldots i])$ が定義されているとき，$\pi[i+1] = f(\pi[0 \ldots i])$ とする．このとき，f' の定義より，$f(\pi[0 \ldots i]) \preceq f'(\pi'[0 \ldots i])$ であるため，$\pi[i+1] \preceq \pi'[i+1]$ である.

    - $f(\pi[0 \ldots i])$ が未定義のとき，$\pi[i+1]$ は，$u \preceq f'(\pi'[0 \ldots i])$ であるような $u \in \pi[i]E$ とする（f' の定義より，このような u は存在する）．このとき，$u \preceq f'(\pi'[0 \ldots i])$ であるため，$\pi[i+1] \preceq \pi'[i+1]$ である.
  - $\pi'[i] \in V_1$ の場合.

    $\pi[i+1]$ は，$u \preceq \pi'[i+1]$ であるような $u \in \pi[i]E$ とする（$\pi'[i+1] \in \pi'[i]E'$ であり，$\pi[i] \preceq \pi'[i]$ であるため，そのような u は存在する）．このとき，$u \preceq \pi'[i+1]$ であるため，$\pi[i+1] \preceq \pi'[i+1]$ である.

□

定 理　4.3. $\mathcal{G} = (V_0, V_1, E, W)$ と $\mathcal{G}' = (V_0', V_1', E', W')$ をそれぞれ *Safety* ゲームとする．また，$\preceq$ を $\mathcal{G}$ と $\mathcal{G}'$ との間の模倣関係とし，$v \preceq v'$ とする．このとき，$\mathcal{G}$ においてプレイヤ *0* が v で必勝であるならば，$\mathcal{G}'$ においてもプレイヤ *0* はノード v' で必勝である.

Proof. $\mathcal{G}$ のノード $v \in V$ においてプレイヤ 0 の必勝戦略 f が存在すると仮定する．定理 4.2 より，$\forall \pi' \in play_{\mathcal{G}'}(f', v').\exists \pi \in play_{\mathcal{G}}(f, v).\pi \preceq_{sq} \pi'$ を満たす $\mathcal{G}'$ の戦略 f' が存在する．f が v でのプレイヤ 0 の必勝戦略であるため，すべての $\pi \in play_{\mathcal{G}}(f, v)$ はプレイヤ 0 の勝利プレイである．π が勝利プレイであるとき，$\pi \preceq_{sq} \pi'$ である π' も，模倣関係の定義より $\pi[i] \in W$ ならば $\pi'[i] \in W'$ であるため，勝利プレイである．したがって，すべての $\pi' \in play_{\mathcal{G}'}(f', v')$ はプレイヤ 0 の勝利プレイであり，f' は，$\mathcal{G}'$ のノード $v' \in V$ においてプレイヤ 0 の必勝戦略である．　□

4.2　模倣関係による無限ゲームの単純化

ここでは，無限ゲームの模倣関係を基にした単純化手法を与える.

ここで与える単純化は，無限ゲームの模倣関係を基に，必勝戦略が存在するかを調べる上で余分なグラフのエッジを除去するものである．具体的には，V_0 に含まれる（V_1 に含まれる）ノードからのエッジに，他の遷移先のノードに模倣される（他の遷移先のノードを模倣する）ノードへのエッジがあった場合，それを除去するものである．

このようなエッジの除去は，同じ模倣関係を基に繰り返し行える．これは，あるゲーム $\mathcal{G}$ の模倣関係 $\preceq$ は，その模倣関係 $\preceq$ を基に $\mathcal{G}$ から余分なエッジを除去したゲーム $\mathcal{G}'$ の模倣関係でもあるためである．

定理 4.4. $\mathcal{G} = (V_0, V_1, E, W)$ をゲームとし，$\preceq$ を G 上の模倣関係とする．また，$e \in E^0 \cup E^1$ であるとし，$E' = E - \{e\}$ をエッジ集合とするゲームを $\mathcal{G}' = (V_0, V_1, E', W)$ とする．ここで，
$E^0 = \{(v,u) \in V^0 \times V^1 | \exists w \in vE.w \neq u \wedge u \preceq w\}$,
$E^1 = \{(v,u) \in V^1 \times V^0 | \exists w \in vE.w \neq u \wedge w \preceq u\}$
である．このとき，以下が成立する．

1. 任意の $v \in V$ で，$\mathcal{G}$ において v でプレイヤ *0* が必勝であるとき，かつそのときにのみ，$\mathcal{G}'$ においても v でプレイヤ *0* は必勝である．
2. $\preceq$ は，$\mathcal{G}'$ 上の模倣関係でもある．

Proof. 1. $\preceq' = \{(v,v) | v \in V\} \cup \preceq$ が，$\mathcal{G}$ と $\mathcal{G}'$ との間の模倣関係であり，かつ，$\mathcal{G}'$ と $\mathcal{G}$ との間の模倣関係であること，および，定理 4.3 より導かれる．ここでは，$\preceq'$ が $\mathcal{G}$ と $\mathcal{G}'$ との間の模倣関係であることのみを示す（$\mathcal{G}'$ と $\mathcal{G}$ との間の模倣関係であることは同様に示される）．

$v \preceq' v'$ であるとする．$\preceq'$ の定義より，明らかに $v \in W$ ならば $v' \in W$ である．

- $v, v' \in V_0$ のとき．$\forall u \in vE.\exists u' \in v'E'.u \preceq' u'$ であることを示す．
  - $v = v'$ のとき．$u \in vE$ とする．$e = (v,u)$ であるとき，$e \in E^0$ であるため，$u \preceq u'$ である $u' \in vE' = v'E'$ が存在する．$e \neq (v,u)$ であるとき，$u \in vE' = v'E'$ であり，$u \preceq' u$ である．
  - $v \preceq v'$ のとき．$u \in vE$ とする．このとき，$v \preceq v'$ であり，$\preceq$ が $\mathcal{G}$ 上の模倣関係あることより，$u \preceq w$ である $w \in v'E$ が存在する．$e = (v',w)$ であるとき，$e \in E^0$ であるため，$w \preceq u'$ である $u' \in v'E'$ が存在し，$u \preceq u'$ である．$e \neq (v',w)$ であるときは，$w \in v'E'$ であり，$u \preceq' w$ である．
- $v, v' \in V_1$ のとき．$\forall u' \in v'E'.\exists u \in vE.u \preceq' u'$ であることを示す．
  - $v = v'$ のとき．$v'E' \subseteq vE$ であることから，明らか．
  - $v \preceq v'$ のとき．$u' \in v'E'$ とする．$E' \subseteq E$ であるため，$u' \in v'E$ である．また，$\preceq$ が $\mathcal{G}$ 上の模倣関係であることより，$u \preceq u'$ である $u \in vE$ が存在し，$u \preceq' u'$ である．

2. $v \preceq v'$ であるとする．

- $v, v' \in V_0$ のとき．$\forall u \in vE'.\exists u' \in v'E'.u \preceq u'$ であることを示す．$u \in vE'$ とする．このとき，$u \in vE' \subseteq vE$ であること，および $\preceq$ が $\mathcal{G}$ 上の模倣関係あることより，$u \preceq w'$ である $w' \in v'E$ が存在する．$e = (v',w')$ であるとき，$e \in E^0$ であるため，$w' \preceq u'$ である $u' \in v'E'$ が存在し，$u \preceq u'$ である．$e \neq (v',w')$ であるときは，$w' \in v'E'$ であり，明らか．
- $v, v' \in V_1$ のとき．上と同様の理由で，$\forall u' \in v'E'.\exists w \in vE'.w \preceq u'$ であることが分かる．

□

5 実現可能性判定におけるゲームの単純化

本章では，実現可能性判定における決定性オートマトン・ゲーム構成において状態・ノードにラベルされる情報より，構成されるゲームの模倣関係が得られることを示し，それを基に，決定性 ω オートマトン・ゲーム構成中に，4.2 節のエッジの除去が可能であることについて述べる．

5.1 3 章の実現可能性判定への適用

3 章で述べた実現可能性判定における決定性オートマトン・ゲーム構成においては，各状態に決定化前のオートマトンの状態から自然数への写像がラベルされる．その自然数は残り何回 F に含まれる状態を通過してよいかを表している．したがって，

$\forall q \in Q.s(q) \leq s'(q)$ である状態 s と s' があった場合，s よりも s' の方がそこでの制約は弱く，s' での方が必勝戦略（実現システム）が存在しやすい．実際，以下に示す通り $\forall q \in Q.s(q) \leq s'(q)$ である場合，s' は s を模倣することができる．

定理 5.1. 前章の実現可能性判定における UC オートマトンを $\mathcal{A} = (2^{X \cup Y}, Q, q_I, \delta, F)$ とし，$\mathcal{D} = (2^{X \cup Y}, S, s_I, \Delta, F^{saf})$ を $\mathcal{A}$ を決定化した *Safety* オートマトンであるとし，$\mathcal{G} = (V_0, V_1, E, W)$ を $\mathcal{D}$ を基に構成される *Safety* ゲームであるとする．

$\preceq^{\mathcal{G}}$ を $R^1 \cup R^0$ とする．ただし，R^1 と R^0 は次のように定める．

$R^1 := \{(s, s') \in S \times S | \forall q \in Q.s(q) \leq s'(q)\}$,

$R^0 := \{(U, U') \in 2^S \times 2^S |$
$\forall u \in U. \exists u' \in U'.(u, u') \in R^1\}$.

このとき，この $\preceq^{\mathcal{G}}$ は，$\mathcal{G}$ 上の模倣関係である．

Proof. $(v, v') \in \preceq^{\mathcal{G}}$ であるとする．定義より，明らかに，$v \in W$ ならば，$v' \in W$ である．

$(v, v') \in V_0 \times V_0$ のとき，つまり，$(v, v') \in 2^S \times 2^S$ のとき，$\preceq^{\mathcal{G}}$ の定義より，$\forall u \in v. \exists u' \in v'.(u, u') \in R^1$ である．$\mathcal{G}$ の定義より，$v = vE$ であるため，$\forall u \in vE. \exists u' \in v'E.(u, u') \in \preceq^{\mathcal{G}}$ である．

続いて，$(v, v') \in V_1 \times V_1$ のとき，つまり，$(v, v') = (s, s') \in S \times S$ の場合を考える．$\mathcal{D}$ の Δ の定義より，$\forall q \in Q.s(q) \leq s'(q)$ ならば，任意の $a \in 2^{X \cup Y}$ における $t = \Delta(s, a), t' = \Delta(s', a)$ において $\forall q \in Q.t(q) \leq t'(q)$ である．これより，任意の $x \in 2^X$ で，$\forall t \in suc(s, x). \exists t' \in suc(s', x).(t, t') \in R^1$ である，つまり，$(suc(s, x), suc(s', x)) \in R_0$ であることが導かれる．したがって，任意の $u' \in s'E$ について（u' は，ある $x \in 2^X$ での $suc(s', x)$），ある $u \in sE$ が存在して（$u = suc(s, x)$），$(u, u') \in \preceq^{\mathcal{G}}$ である． □

このようにゲーム構成時においてノードにラベルされる情報より，構成されるゲームの模倣関係 $\preceq^{\mathcal{G}}$ が得られる．それを基に，$v \in V_0$ においては，$\exists w \in vE.w \neq u \wedge u \preceq w$ である $(v, u) \in E$ を，$v \in V_1$ においては，$\exists w \in vE.w \neq u \wedge w \preceq u$ である $(v, u) \in vE$ を除去できる（定理 4.4 より，このようなエッジの除去は実現可能性判定の結果に影響を与えない）．このようなエッジ除去をゲーム構成時に順次行うことで，初期ノードから到達可能な部分，すなわち，ゲームを解くのに必要なノードやエッジが減る．決定性オートマトンも，ゲームのグラフを構成するのに必要な部分のみを順次構成していくことで，同様に構成しなければならない状態が減る．したがって，決定性オートマトン・ゲームのグラフを構成するコスト，ゲームを解くコストの両方が削減され，実現可能性をより効率的に行えるようになる．

5.2 他の実現可能判定法への適用

前節では，3 章で述べた実現可能性判定での決定性オートマトン・ゲーム構成において，4.2 節のエッジ除去を適用する方法について述べたが，他の決定性オートマトン・ゲーム構成においても，同様に適用可能である．例えば，[16] [17] の LTL フラグメントの実現可能性判定法での決定性オートマトン・ゲーム構成は部分集合構成法を基するため，そこでの決定性オートマトン・ゲームの各状態には決定化前の状態の集合がラベルされる．その状態集合の包含関係を基に模倣関係が定義できる．それを利用し，同様な単純化が適用可能である．

5.3 本単純化の特徴

4.2 節のゲームの単純化は必勝戦略が存在するかを変化させないものである．実現可能性におけるゲームの必勝戦略は検証対象の仕様の実現システムと対応するため，ここでの単純化は「実現システムが存在するか」を変化させない単純化であるといえる．しかし，「それが表現する実現システム集合」は変化させている．例えば，システムの応答に対応する 0-ノードからのエッジの除去は，ある応答後の状態が他の応答後の状態に模倣される場合，そのような応答を行うシステムを排除することに対応する．これは，[11] 等の実現可能性判定の効率化のための単純化と大きく異なる点である．「実現システムが存在するか」のみを変化させないという，緩い条件を基に単純化を行うため，ここでの単純化は強力である．

6 実験

我々は，本稿で与えた無限ゲームの単純化をゲームのグラフを構成時において適用することで，どれだけ実現可能性判定のコストを削減できるかを調査する実験を行った．本章では，その結果について述べる．

6.1 実験内容

以下の実現可能性判定法に対し，本稿で与えた無限ゲームの単純化を適用することの効果を調査した．

- 3 章で述べた実現可能性判定法.
- [16][17] の LTL フラグメントの実現可能性判定法.

本実験では，構成されるゲームのグラフのノード数，グラフの構成時間と判定にかかる総時間を測定し，単純化によりそれらがどのように変化するかを調べる．

本実験で検証対象とする仕様は，付録に示す n プロセス排他制御の仕様，及び簡易 n 階建てエレベータの仕様である．

本実験で利用する実現可能性判定法の実装は，[3][4] の BDD（Binary Decision Diagram）や MTBDD（multi-terminal BDD）を用いたタブロー法（非決定性 ω オートマトンの構成法）やオートマトンの決定化の実装法を基に，作成したものである．実装言語は OCaml であり，BDD 操作に関する部分は CUDD(Colorado University Decision Diagram package)[18] を利用している．

実験に利用する計算機環境は，CPU: Intel(R) Pentium(R) D 3.0GHz，メモリ: 2.0GB，OS: openSUSE 11.0 である．

6.2 結果

3 章で述べた実現可能性判定法に対するベンチマーク結果を表 1 と表 2 に示す．また，[16][17] の LTL フラグメントの実現可能性判定法に対するベンチマーク結果を表 3 と表 4 に示す．ここで，“> 1800” は 1800 秒経過しても判定が終了しなかったことを表す．なお，判定結果は，いずれも「実現可能である」であった．

表 1 と表 2 より，3 章の実現可能性判定法に対し，本稿で与えた単純化手法を適用することで，グラフのノード数，グラフの構成時間と総判定時間がそれぞれいずれの仕様の判定においても減少していることがわかる．n 階建てエレベータの仕様では，$n = 2$ のとき単純化を行わない場合ノード数は 56370 であるが単純化を行うことで半分以下の 22085 にまで減少している．グラフ構成時間と総判定時間は，単純化のオーバーヘッドがあるためそこまでではないが，2 割弱削減されている．n プロセス排他制御の仕様では，より単純化が効いており，例えば $n = 6$ のとき，単純化を行わない場合ノード数は 782599 であるのに対し，単純化を行う場合ノード数は 83587 であり，約 1 割ほどに抑えられている．それに伴い，総判定時間も 368.91 秒から 14.26 秒へと大きく削減されている．参考のため，他の実現可能性判定器との判定時間の比較も行った．構文制限のない LTL 仕様の実現可能性判定ツールとしては，我々の知る限り，Lily[11] と Acacia[7] しか存在しない．それらでの判定時間を表 5 と表 6（単位 : 秒）に示す．いずれの仕様の判定でも本研究の実装が最速であった（本稿で与えた単純化を適用させなくとも）．

表 3 と表 4 より，[16][17] の LTL フラグメントの実現可能性判定法についても同様に，本稿で与えた単純化手法を適用することで，グラフのノード数，グラ

表 5 他のツールとの判定時間の比較

（n プロセス排他制御の仕様）

	本実装		Lily	Acacia
n	単純化あり	単純化なし		
4	0.51	0.41	5.99	1048.85
5	4.60	1.20	85.99	>1800
6	369.21	14.26	1445.67	
7	>1800	515.15	>1800	

表 6 他のツールとの判定時間の比較

（簡易 n 階建てエレベータ仕様）

	本実装		Lily	Acacia
n	単純化あり	単純化なし		
2	125.63	102.79	>1800	439.67
3	>1800	>1800		>1800

表 1　3 章の実現可能性判定法への単純化適用の効果（n プロセス排他制御の仕様）

	単純化なし			単純化あり		
n	ノード数	グラフ構成時間（秒）	判定時間（秒）	ノード数	グラフ構成時間（秒）	判定時間（秒）
2	59	0.24	0.29	49	0.24	0.29
3	409	0.28	0.33	171	0.28	0.33
4	4049	0.44	0.51	920	0.36	0.41
5	41127	3.79	4.60	9031	1.11	1.20
6	782599	285.79	369.91	83587	13.41	14.26
7			>1800	873080	469.37	515.15
8						>1800

表 2　3 章の実現可能性判定法への単純化適用の効果（簡易 n 階建てエレベータの仕様）

	単純化なし			単純化あり		
n	ノード数	グラフ構成時間（秒）	判定時間（秒）	ノード数	グラフ構成時間（秒）	判定時間（秒）
2	56370	124.42	125.63	22085	102.55	102.79
3			> 1800			> 1800

表 3　[16] [17] の実現可能性判定法への単純化適用の効果（n プロセス排他制御の仕様）

	単純化なし			単純化あり		
n	ノード数	グラフ構成時間（秒）	判定時間（秒）	ノード数	グラフ構成時間（秒）	判定時間（秒）
6	129	0.04	0.04	8	0.03	0.03
7	257	0.06	0.07	9	0.03	0.04
8	513	0.11	0.16	10	0.04	0.04
9	1025	0.24	0.39	11	0.05	0.05
10	2049	0.71	1.19	12	0.08	0.08
11	4087	2.29	3.94	13	0.12	0.14
12	8193	28.50	34.18	14	0.24	0.26
13	16385	77.01	96.18	15	0.52	0.53
14	32768	937.63	1012.31	16	1.16	1.17
15			>1800	17	2.64	2.66
16				18	6.04	6.05
17				19	15.33	15.38
18				20	52.75	52.82

フの構成時間と総判定時間がそれぞれ減少していることがわかる．特に，n プロセス排他制御の仕様では，その効果は大きく，単純化を適用することで，ノード数が $n+2$ に抑えられている．それにより，総判定時間は，例えば $n = 16$ の場合 1012.31 秒から 1.17 秒へと，大きく削減されている．

以上のように，本稿で与えた単純化によって，より効率的に実現可能性判定を行えるようになることが確認された．

7　関連研究

[5] [6] [8] では無限語上のオートマトンの単純化手

表 4 [16] [17] の実現可能性判定法への単純化適用の効果（簡易 n 階建てエレベータ仕様）

	単純化なし			単純化あり		
n	ノード数	グラフ構成時間（秒）	判定時間（秒）	ノード数	グラフ構成時間（秒）	判定時間（秒）
2	73	0.03	0.04	47	0.03	0.04
3	495	0.10	0.31	271	0.05	0.11
4	4568	1.30	8.75	1630	0.37	2.21
5	44697	83.05	303.21	10348	5.03	34.87
6			>1800	66691	213.32	644.00
7						>1800

法が，[2] では有限木上のオートマトンの単純化手法が，そして，[11] では無限木上のオートマトンの単純化手法がそれぞれ提案されている．

それらは，いずれもオートマトンの模倣関係を基に単純化を行う．特に [8] [11] では，オートマトン構成時に状態にラベルされる情報から得られる模倣関係を基に単純化を行う．本研究の単純化は，その点ではそれらの単純化と同様である．

しかし，それらはいずれも受理語集合や受理木集合を変化させない単純化である．[11] では，実現可能性判定効率化のための無限木上のオートマトンの単純化手法が提案されているが，それは，「それが表現する実現システム集合」を変化させない単純化であるといえる．本研究の単純化は，「実現システムが存在するか」のみを変化させないというより緩い条件を基にした単純化であり，その点がそれら単純化と大きく異なる．このような緩い条件を基にするため，本研究の単純化はより強力である．

8 まとめ

本研究では，実現可能性判定コスト削減のための無限ゲームの単純化手法を与えた．ここで与えた単純化は，無限ゲームの模倣関係を基に，必勝戦略が存在するかを調べる上で余分な無限ゲームのグラフのエッジを除去するものである．この単純化は，無限ゲームの構成途中において適用することが可能であるため，無限ゲームの構成自体のコストをも下げることができる．実現可能性判定の効率化のための既存の単純化との違いは，「それが表現する実現システム集合」を変化させいないのではなく，「実現システムが存在するか」のみを変化させないというより緩い条件を基にした単純化であるという点である．

我々は，本稿で与えた単純化の効果を調査する実験も行い，本単純化を適用することで，より効率的に実現可能性判定を行えることを確かめた．適用する実現可能性判定法や検証対象の仕様によっては，その効果は絶大であり，例えば，[16] [17] の LTL フラグメントの実現可能性判定法に対して，本単純化を適用させた場合，付録の n プロセス排他制御の仕様の判定では，$n = 14$ のとき，構成されるグラフのノード数は 32768 から 16 へと，判定時間は 1012.31 秒から 1.17 秒へと大きく減少した．

本研究で与えた単純化は，それを行っても，仕様が実現可能であると判定された場合，有限状態機械として表現される実現システムを合成することが可能であるため，プログラム合成 [13] の効率化にも有効である．与えられたシステムが仕様を満たすかを検証するモデル検査などと比べ，仕様から自動的に正しく動作することが保証されたシステムを合成するプログラム合成は，現状では計算コストの高さから小規模な対象にその適用が限られている．我々は今後，本研究の成果を基に，プログラム合成のより大規模で実際的な問題への適用に挑戦していく予定である．

参 考 文 献

[1] Abadi, M., Lamport, L., and Wolper, P.: Realizable and Unrealizable Specifications of Reactive Systems, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, 1989, pp. 1–17.

[2] Abdulla, P. A., Bouajjani, A., Holík, L., Kaati, L., and Vojnar, T.: Computing simulations over

tree automata: efficient techniques for reducing tree automata, *TACAS'08/ETAPS'08: Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, Berlin, Heidelberg, Springer-Verlag, 2008, pp. 93–108.

[3] 青島武信: リアクティブシステム仕様の検証系に関する研究, 博士論文, 東京工業大学大学院情報理工学研究科 計算工学専攻, 2002.

[4] 青島武信, 米崎直樹: 時間論理によるリアクティブシステム仕様の検証の効率化, コンピュータソフトウェア, Vol. 20(2003), pp. 30–53.

[5] Etessami, K. and Holzmann, G. J.: Optimizing Büchi Automata, *CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory*, London, UK, Springer-Verlag, 2000, pp. 153–167.

[6] Etessami, K., Wilke, T., and Schuller, R. A.: Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata, *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, London, UK, Springer-Verlag, 2001, pp. 694–707.

[7] Filiot, E., Jin, N., and Raskin, J.-F.: An Antichain Algorithm for LTL Realizability, *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, Berlin, Heidelberg, Springer-Verlag, 2009, pp. 263–277.

[8] Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata, *CIAA'03: Proceedings of the 8th international conference on Implementation and application of automata*, Berlin, Heidelberg, Springer-Verlag, 2003, pp. 35–48.

[9] Gerth, R., Peled, D., Vardi, M. Y., and Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic, *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, London, UK, UK, Chapman & Hall, Ltd., 1996, pp. 3–18.

[10] Jackson, D.: Automating first-order relational logic, *Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering*, 2000, pp. 130–139.

[11] Jobstmann, B. and Bloem, R.: Optimizations for LTL Synthesis, *Formal Methods in Computer Aided Design, 2006. FMCAD '06*, Nov. 2006, pp. 117–124.

[12] Kupferman, O. and Vardi, M. Y.: Safraless Decision Procedures, *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, Washington, DC, USA, IEEE Computer Society, 2005, pp. 531–542.

[13] Pnueli, A. and Rosner, R.: On the synthesis of a reactive module, *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1989, pp. 179–190.

[14] Safra, S.: On the complexity of omega-automata, *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1988, pp. 319–327.

[15] Schewe, S. and Finkbeiner, B.: Bounded Synthesis, *Proc. ATVA*, 2007, pp. 474–488.

[16] 島川昌也, 萩原茂樹, 米崎直樹: 仕様の自動検証に適した LTL フラグメント–実現集合を表す決定性オートマトン構成の立場から–, 日本ソフトウェア科学会第 *23* 回大会講演論文集, 2006.

[17] 島川昌也, 萩原茂樹, 米崎直樹: 実現可能性判定コスト削減のための LTL 構文の制限とそれによる仕様の判定法, 日本ソフトウェア科学会第 *26* 回大会講演論文集, 2009.

[18] Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.2.0, 1998.

A ベンチマークに用いた仕様

ここでは，ベンチマークに用いた仕様を示す.

n プロセス排他制御の仕様

n プロセス排他制御の仕様を以下に示す．ここで，X は要求イベントの生起を表す原子命題の集合, Y は応答イベントの生起を表す原子命題の集合である.

$X = \{$
$req[i](i = 1..n)$. //プロセス i がリソース要求を出した
$\}$
$Y = \{$
$serv[i](i = 1..n)$. //プロセス i に対してリソースを割り当てている
$\}$

$$\mathbf{G}(\bigwedge_{1\le i\le n}(req[i] \rightarrow \mathbf{F}serv[i])) \quad \wedge$$
$$\mathbf{G}(\bigwedge_{1\le i\le n}(serv[i] \rightarrow \bigwedge_{1\le j\le n, j\neq i}\neg serv[j])).$$

簡易 n 階建てエレベータの仕様

簡易 n 階建てエレベータの仕様は,

$$\bigwedge \Phi \ \rightarrow \ \bigwedge \Psi$$

であるとする．Φ と Ψ は以下に示す式集合である.

$X = \{$
$ReqBtn[i](i = 1..n)$, //i 階の呼出しボタンが押された
$OpenBtn$, //「開く」ボタンが押された
$CloseBtn$ //「閉じる」ボタンが押された

$\}$
$Y = \{$
$Loc[i](i = 1..n)$, //リフトが i 階にある
$ReqL[i](i = 1..n)$, //リフトが i 階に行く要求がある
$Open$, //ドアが開いている
$Move$, //リフトが可動状態である
$OpenTimedOut$, //「開く」ボタンの時間切れ
$OpenReq$ //ドアを開く要求がある
$\}$

$\Phi = \{$

$\bigwedge_{1 \le i \le n} \mathbf{GF} \neg ReqBtn[i]$

$\}$

$\Psi = \{$

//リフトはどこかの階にある

$\mathbf{G}(\bigvee_{1 \le i \le n} Loc[i])$,

//リフトがある階にある場合，他の階にはない

$\mathbf{G}(\bigwedge_{1 \le i \le n} (Loc[i] \rightarrow \bigwedge_{j=1..n, i \neq j} \neg Loc[j]))$,

//呼び出しボタンが押されればその階にいつか行く

//かつその要求が満たされるまで要求し続ける

$\mathbf{G}(\bigwedge_{1 \le i \le n} (ReqBtn[i] \rightarrow \mathbf{F} Loc[i] \wedge ReqL[i] \ \mathbf{U}_w (Loc[i] \wedge ReqL[i])))$,

//リフトのある階に要求があればドアを開く

//ドアが開いている間，リフトは停止

$\mathbf{G}(\bigwedge_{1 \le i \le n} (Loc[i] \wedge ReqL[i] \rightarrow Open \wedge Loc[i] \ \mathbf{U}_w Move))$,

//呼び出しボタンが押されるまで要求は出さない

$\mathbf{G}(\bigwedge_{1 \le i \le n} (Loc[i] \wedge Move \rightarrow \neg ReqL[i] \ \mathbf{U}_w ReqBtn[i])$,

//その階に要求がなければ，ドアは開かない

$\mathbf{G}(\bigwedge_{1 \le i \le n} (Loc[i] \wedge \neg ReqL[i] \rightarrow \neg Open))$,

//途中階を通る

($n \ge 3$ の場合)

$\mathbf{G}(\bigwedge_{\substack{1 \le i \le n-2 \\ 3 \le j \le n \\ i \le j-2}} (Loc[i] \wedge ReqL[j] \rightarrow \bigwedge_{i+2 \le k \le j} Loc[k-1] \mathbf{R}_s \neg Loc[k]))$,

($n \ge 3$ の場合)

$\mathbf{G}(\bigwedge_{\substack{1 \le i \le n-2 \\ 3 \le j \le n \\ i \le j-2}} (Loc[j] \wedge ReqL[i] \rightarrow \bigwedge_{i+2 \le k \le j} Loc[k+1] \mathbf{R}_s \neg Loc[k]))$,

//ドアの開閉とリフトの可動状態の関係

$\mathbf{G}(Open \rightarrow \neg Move \ \mathbf{U}_w \neg Open)$,

$\mathbf{G}(\neg Open \rightarrow Move \ \mathbf{U}_w Open)$,

//ドアの開放状態には時間制限がある

$\mathbf{G}(Open \rightarrow \mathbf{F} OpenTimedOut)$,

//時間制限内で “開く” ボタンを押せば

//ドアの開放を要求

$\mathbf{G}(OpenBtn \wedge \neg OpenTimedOut \rightarrow OpenReq)$,

//ドア開放時間制限が過ぎたらドアを閉める

$\mathbf{G}(OpenTimedOut \rightarrow \neg Open)$,

//開放要求がなくかつ “閉じる” ボタンが押されれば

//ドアを閉じる

$\mathbf{G}(CloseBtn \wedge \neg OpenReq \rightarrow \neg Open)$,

//ドア開放要求が真，可動状態が偽ならドアを開く

$\mathbf{G}(OpenReq \wedge \neg Move \rightarrow Open)$

$\}$

ただし，演算子 $\mathbf{U}_w$, $\mathbf{R}_s$ は，以下のような略記とする．

$f_1 \ \mathbf{U}_w \ f_2 \equiv f_2 \ \mathbf{R}(f_2 \vee f_1)$ ［弱い until］,

$f_1 \ \mathbf{R}_s \ f_2 \equiv f_2 \ \mathbf{U}(f_2 \wedge f1)$ ［強い release］.